

REPORT DOCUMENTATION PAGE*Form Approved*
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Service, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington, DC 20503.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**1. REPORT DATE (DD-MM-YYYY)**

AUGUST 2010

2. REPORT TYPE

Conference Paper

3. DATES COVERED (From - To)

May 2008 – March 2010

4. TITLE AND SUBTITLEIMPROVING COMPLEX DISTRIBUTED SOFTWARE SYSTEM
AVAILABILITY THROUGH INFORMATION HIDING**5a. CONTRACT NUMBER**

In House

5b. GRANT NUMBER

N/A

5c. PROGRAM ELEMENT NUMBER

62702F

6. AUTHOR(S)

Li Wang, Yair Leiferman, Shangping Ren, Kevin Kwiatt, and Xiaowei Li

5d. PROJECT NUMBER

23G4

5e. TASK NUMBER

IH

5f. WORK UNIT NUMBER

01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

AFRL/RIGG	Illinois Institute of Technology	Institute of Computing Technology
525 Brooks Road	Department of Computer Science	Chinese Academy of Sciences
Rome NY 13441-4505	Chicago, IL 60616	Beijing, China 100080

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)AFRL/RIGG
525 Brooks Road
Rome NY 13441-4505**10. SPONSOR/MONITOR'S ACRONYM(S)**
N/A**11. SPONSORING/MONITORING AGENCY REPORT NUMBER**
AFRL-RI-RS-TP-2010-28**12. DISTRIBUTION AVAILABILITY STATEMENT**

Approved For Public Release; Distribution Unlimited. PA #: 88ABW-2009-2758

Date Cleared: 23-Jun-09

13. SUPPLEMENTARY NOTES

© 2010 ACM. This paper was published in the Proceedings of the 25th ACM Symposium on Applied Computing. This work is copyrighted. One or more of the authors is a U.S. Government employee working within the scope of their Government job; therefore, the U.S. Government is joint owner of the work and has the right to copy, distribute, and use the work. All other rights are reserved by the copyright owner.

14. ABSTRACT

For distributed software systems, ensuring their availability under intentional attacks is critical. Traffic analysis, conducted by the attacker, could reveal the protocol being carried out by the components. Furthermore, having inferred the protocol, the attacker can use the pattern of the messages as a guide to the most critical components. These directed attacks can be thwarted by using message forwarding at the application level to reduce traffic differences, thus diverting attackers from targeted attack to random attack, which probabilistically prolongs the availability of important components in the system. The simulations results also show that message forwarding effectively balance the traffic flow and hence indicate the validity of the approach.

15. SUBJECT TERMS

Reliability Message Forwarding, Availability, Complex Distributed Software

16. SECURITY CLASSIFICATION OF:**17. LIMITATION OF ABSTRACT****18. NUMBER OF PAGES****19a. NAME OF RESPONSIBLE PERSON**

Kevin A. Kwiatt

a. REPORT

U

b. ABSTRACT

U

c. THIS PAGE

U

UU

6

19b. TELEPHONE NUMBER (Include area code)

N/A

Improving Complex Distributed Software System Availability Through Information Hiding

Li Wang, Yair Leiferman,
Shangping Ren^{*}
Department of CS
Illinois Institute of Technology
Chicago, Illinois 60616
{lwang64,yleiferm,ren}@iit.edu

Kevin Kwiat[†]
Cyber Science Branch
Air Force Research
Laboratory, AFRL/RIGG
Rome, NY 13441-4505
kwiatk@rl.af.mil

Xiaowei Li
Institute of Computing
Technology
Chinese Academy of Sciences
Beijing, China 100080
lxw@ict.ac.cn

ABSTRACT

For distributed software systems, ensuring their availability under intentional attacks is critical. Traffic analysis, conducted by the attacker, could reveal the protocol being carried out by the components. Furthermore, having inferred the protocol, the attacker can use the pattern of the messages as a guide to the most critical components. We thwart these directed attacks by using message forwarding to reduce traffic differences, thus diverge attackers from targeted attack to random attack, which probabilistically prolongs the availability of important components in the system. The simulation results also show that message forwarding effectively balance the traffic flow and hence indicate the validity of our approach.

Categories and Subject Descriptors

C.4 [PERFORMANCE OF SYSTEMS]: Reliability, Availability, and Serviceability; D.4.6 [OPERATING SYSTEMS]: Security and Protection—*Information flow controls*

General Terms

Reliability

Keywords

Message Forwarding, Availability, Traffic Analysis, Complex Distributed Software System

1. INTRODUCTION

^{*}The work is supported by NSF CAREER Award (CNS0746643)

[†]Approved for Public Release; distribution unlimited: 88ABW-2009-2758

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'10 March 22-26, 2010, Sierre, Switzerland.

Copyright 2010 ACM 978-1-60558-638-0/10/03 ...\$10.00.

Our society is increasingly dependent on distributed software systems which, on the other hand, often become the targets of malicious attacks. In order to survive against intentional attack, as pointed out by Li et al. [15], we should prevent attackers from learning “secret” information about the systems. Hence, creating an elusive target has become a research thrust. Frequency hopping [2] and n-version programming [5, 3] are examples of using known approaches to address the increasingly hostile environment created by attackers. Research in complex network systems’ vulnerability to attacks [21, 1, 6, 4] also show that if certain amount of information about the network is hidden, the system’s survivability against intentional attacks can be greatly improved [20].

Traffic pattern within a network often reveal different components’ characteristics [22] in a system. If such information is not protected, attackers can easily figure out the locations of more important components through traffic analysis, irrespective to whether important components have more traffic or not. For instance, a log server component is relative less important, but receives more messages than other components in the system; whereas in the 2-phase commit protocol there are many cohorts but only one coordinator which is the critical node and has more communication in the system.

To render attackers’ traffic analysis efforts less effective, the technique of traffic padding [22, 8] is used to disguise a network communications protocol. It creates a uniform level of traffic, so an attacker cannot discern what protocol is in use and hence exploit the protocol’s vulnerabilities. However, complex distributed software systems are often modeled and built by distributed components that communicate with each other through messages [19, 16]. For systems that are reconfigurable (to obscure the system from attack), the deployment of components in a distributed system is not constant. Hence, it is not possible to know, a priori, what communications protocol the links between components can support. Traffic padding does not support this granularity of control. We hence need to raise traffic padding from a networking technique to one that is more appropriate for a distributed system whose designers employ multiple levels of system abstraction to handle the system’s complexities.

The rest of the paper is organized as follows. Section 2 discusses related work. The detailed discussion about applying information hiding principle to improve complex

distributed software systems’ availability is presented in Section 3. Section 4 discusses simulation and experiment results. We conclude and point out future work in Section 5.

2. RELATED WORK

In order for a distributed system executing in an open environment to survive from attacks, a straightforward and commonly used approach is to use redundancy. If a primary component is compromised, the backup spares take over the work and maintain the availability of the system. However, this solution not only increases the cost of the system, but also over time the redundancy itself may contribute to the decrease of system dependability [18].

Levitin et al. [14] analyze how to maintain system availability with fixed resources despite the presence of external attacks. In their model, a system is composed of identical elements and the defender uses redundancy by deploying more genuine elements than needed. The defender may use the resource budget to strengthen the protection of the individual genuine elements or deploy false targets (i.e., decoys) to lure the attacker away from genuine elements. Optimal resource allocation strategies that cope with different attack situations are then analyzed. Hausken et al. [11] further explicate an optimal distribution of fixed resources to maximize availability of series systems through the protection of individual genuine elements and deployment of false targets. In [9, 10] the optimal defense resource allocation is determined in order to maximize availability of homogeneous parallel system from attacks through the combination of protection and redundancy or false targets and redundancy.

Our approach differs from the aforementioned research in two aspects. First, we extend system’s availability without introducing new components, such as enhanced protection, replicas, etc., into the systems. Second, we do not assume software components are homogeneous. Rather, we focus on the diversities of components in the system.

Onion Routing [7] is used to resist eavesdropping, traffic analysis and other attacks both from outsiders and insiders. It is a good approach to prevent attackers from getting to know about the system’s network topology. Our approach goes beyond Onion Routing in the sense that when attackers have successfully figured out the network topology, we can still protect the system by prolonging the time the attackers need to compromise the system.

In [12], we propose a coordination model to improve software system attack-tolerance and survivability in open hostile environment. Survivable feedback loops are built in this distributed coordination model to exclude the faulty entities from the system and protect the system from being broken down by single failures. However, our earlier work focuses on expelling faulty components from the system in order to eliminate their anomalous behaviors and thereby improve system survivability. While our current approach aims at preventing attackers from quickly identifying and attacking the core components rather than tolerating the faults caused by the attack.

3. APPLY INFORMATION HIDING TO IMPROVE COMPLEX DISTRIBUTED SOFTWARE SYSTEMS’ AVAILABILITY

In this section, we first define the system model and state the assumptions to be used in the paper. We then formulate complex distributed software systems’ availability problem and provide our solutions.

3.1 System Model and Assumptions

A complex distributed software system is modeled as a set of asynchronous, autonomous, and heterogeneous software components that communicate with each other through messages. We assume that not all components are equally important and the amount of communications flowing through different components are different. There is a small set of components that are critical to the functioning of the system. These components are called core components. The physical nodes which contain core software components are called core nodes. The traffic flow is generated when communicating components are on different physical nodes. Compromising non-core nodes only degrades the system’s performance, while the system loses its availability when all core nodes are comprised. We further assume that if a physical node is compromised, all the components that reside on the node are compromised.

An attacker is a malicious entity whose goal is to compromise the system. We assume that attackers know locations and connection topology of the physical nodes, but initially have no information regarding which software component(s) is or are deployed on a specific physical node. We further assume that the only information that is accessible to the attackers is system’s network traffic flow, however, they cannot decipher message contents carried by the communication flow.

Our main design objective is to maximize the time required by attackers to comprise the system and at the same time minimize the extra cost added to the system.

3.2 Defending Approach

Laprie [13] defines system’s availability as its readiness for providing its correct service. In this paper, we relate system’s availability *improvement* to the prolonged time that an attacker needs to comprise the system.

As stated in earlier, in our model, not all components are equally critical with respect to the system’s availability. Therefore, the attacker who wants to succeed but not spend too much time launching the attack (in order to escape detection) will locate and target the physical nodes that contains the system’s important or critical components. When the system starts, attackers may have no or little information regarding where different software components are deployed. However, as time goes on, the attackers will gather sufficient information from the communications among components that leads them to the critical nodes where the system’s core components reside. They make this judgment based on the analysis of traffic pattern. However, the precision with which an attacker can analyze the traffic pattern determines the amount of information that he can obtain from analysis [17].

To decrease the attacker’s traffic pattern analysis precision, we camouflage the real traffic flow among components. In particular, we forward messages among components so that the traffic flow among all the components appears more similar and lure attackers away from correct traffic pattern analysis. Although it is impossible to stop attackers’ criminal intention of comprising a system, if we are able to change

attacker's informed and targeted attack to random attack, we effectively prolong the system system's availability.

Clearly, using message forwarding is not the only way to hide the real traffic flow. For instance, we can create a dummy node that generate false traffics. However, with dummy node approach, the dummy node itself could become a single point of failure for the defending scheme — the attacker will easily discover all core nodes once the dummy node is identified; while adding more dummy nodes increases resource costs and degrades system's performance. Message forwarding overcomes these shortcomings. It is easy to implement and does not increase much resource need.

Assume there are total number of N software components in the system and the allowed maximal number forwards is f . For a message msg with destination component $dest$, Algorithm 1 defines the process of message forwarding. The algorithm is executed when a component receives a message.

Algorithm 1 MESSAGE FORWARDING($N, f, dest, msg$)

```

1: for  $i \leftarrow 1$  to  $f$  do
2:   randomly generate a number  $j$  within  $[1, N]$  except
   the sender ID
3:   if  $component_j = dest$  then
4:     dispatch  $msg$  to destination component  $dest$ 
5:   return
6:   else
7:     forward  $msg$  to  $component_j$ 
8:   end if
9: end for
10: dispatch  $msg$  to destination component  $dest$ 
11: return

```

Clearly, message forwarding adds communication cost to the system. The larger the allowed maximal number of forwards, the more redundant messages are added to the system. To control the communication overhead generated by the message forwarding, we have to constrain the maximal number of forwards to bound the overhead. Hence, the question arises in deciding the optimal number of forwards needed to sufficiently hide the real traffic pattern from being discovered by attackers while maintaining the minimal communication overhead.

3.3 Deciding Optimal Number of Forwards

Assume there are N components in the system. If the maximal number of forwards is 0, messages are directly sent to their destination components. Suppose without forwarding, the number of messages each component sends and receives is $msg_{j,0}$, where $1 \leq j \leq N$, and the number of total messages communicated in the system is $MSG_0 = \sum_{j=1}^N msg_{j,0}$. When the maximal number of forwards is f , as Algorithm 1 (line 2 to line 8) indicates that it is possible that message reaches its destination without being forwarded, or after traveling through some forwarding nodes. When the message is forwarded, we assume the probability that each component except the sender itself receives the message is uniformly distributed. We use $msg_{j,f}$ to denote the number of messages going through component j when f number of forwards is allowed, where $1 \leq j \leq N$, hence, the number of total messages in the system is $MSG_f = \sum_{j=1}^N msg_{j,f}$. In addition, we maintain the traffic flow difference below a threshold α .

When the maximal number of forwards is f , there are $f + 1$ outcomes that a message reaches destination component from original sender according to Algorithm 1. The probability of each outcome is listed as follows:

Case 1: The probability that a message reaches its destination component through i forwards, where $0 \leq i < f$, and f is the maximal number of forwards allowed:

$$p_{0 \leq i < f} = \left(\frac{N-2}{N-1}\right)^i \times \frac{1}{N-1} \quad (1)$$

Case 2: The probability that a message does not reach its destination component through f forwards, rather it is forced to reach its destination according to line 10 in Algorithm 1.

$$p_{i=f} = \left(\frac{N-2}{N-1}\right)^f \quad (2)$$

When a message is forwarded i times, there are totally $i + 1$ messages. Therefore, for a given number of maximal forwards f , the expected number of total messages is:

$$\begin{aligned}
E(X = f) &= \sum_{i=0}^{f-1} (i+1) \times p_{0 \leq i < f} + (f+1) \times p_{i=f} \\
&= \sum_{i=1}^f \frac{i}{N-1} \times \left(\frac{N-2}{N-1}\right)^{i-1} + (f+1) \times \left(\frac{N-2}{N-1}\right)^f
\end{aligned} \quad (3)$$

Hence, the total number of messages communicated within the systems is

$$MSG_f = MSG_0 \times E(X = f) \quad (4)$$

For a component j , the total messages that can be forwarded are all the messages in the system except the ones that have itself as the destination. Hence, on average, number of messages added to component j is

$$\frac{1}{N-1} \times (E(X = f) - 1) \times (MSG_0 - msg_{j,0})$$

Hence, the total number of messages sent and received by a component j is $msg_{j,f}$ =

$$msg_{j,0} + \frac{1}{N-1} \times (E(X = f) - 1) \times (MSG_0 - msg_{j,0}) \quad (5)$$

Assume component h has the most traffic, while component l has the least traffic in the system, and we want to maintain the traffic flow difference is below the threshold α to avoid attacker from quickly identifying the traffic pattern, i.e.,

$$\frac{msg_{h,f} - msg_{l,f}}{MSG_f} \leq \alpha \quad (6)$$

Therefore, in order to hide core components from being discovered through traffic analysis, based on the (4), (5), and (6), we shall ensure (7).

$$E(X = f) \geq \frac{(msg_{h,0} - msg_{l,0}) \times N}{(msg_{h,0} - msg_{l,0}) + \alpha \times (N-1) \times MSG_0} \quad (7)$$

The following example illustrates the process of deciding minimal number of maximal forwards for a distributed software system.

Example 1: Assume a distributed system has 10 components. Without message forwarding, the total traffic in

the system is 1000 messages. There are two components which have more traffic flow than the other components, and each of these two component has 40% of the total number of messages, and the other software components share the rest of the messages averagely, which is 2.5% of the total messages. The traffic different threshold α is set to be 5%. We are to decide the minimum f so that the traffic difference is not above the threshold. Solving (7) with the data above, we get

$$E(X = f) \geq 4.55 \quad (8)$$

From (8), we know that if the expected number of messages is larger than 4.55, the traffic flow difference among different nodes are below the predefined threshold. Based on Equation (3), we have

$$\begin{aligned} E(X = 0) &= 1.00; & E(X = 1) &= 1.89; \\ E(X = 2) &= 2.68; & E(X = 3) &= 3.38; \\ E(X = 4) &= 4.01; & E(X = 5) &= 4.56; \end{aligned}$$

It is easy to see that Equation (3) is a monotonically increasing function. As can be seen from the above calculation, the optimal value of maximal forwards is 5.

4. SIMULATION RESULTS

In this section, we discuss the simulation results. The purpose of the simulations is to investigate the relationship between the number of forwards and traffic difference reduction among components.

In this experiment, there are ten components in the system, and two of them have high traffic flow. In particular, we assume that the initial total number of messages without forwarding is 1000, and each of the two high traffic components has 40% of the total messages, and the rest components evenly receive the last 20% of the total messages, which is 2.5% of total messages on each low traffic components. When a component receives or sends a message, it increases the number of messages.

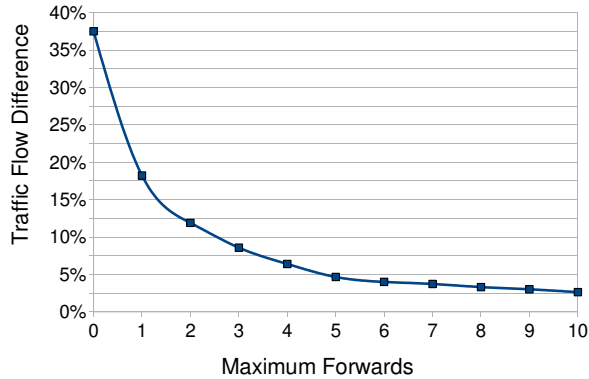


Figure 1: Traffic difference deduction based on the maximal number of forwards

We set the maximal number of forwards from 0 to 10 and Figure 1 shows the traffic difference reduction between highest traffic component and lowest traffic component. As shown in Figure 1, when the number of allowed message forwarding increases, the traffic difference decreases to around 18% with one forwarding, and reaches below $\alpha =$

5% when the maximal number of forwards is 5. The figure also indicates that when the allowed number of forwards increases beyond 5, the traffic difference does not decrease much, Hence, under the test setting, the optimal number of forwards is 5 which is consistent with the analysis given in Example 1 in Section 3.

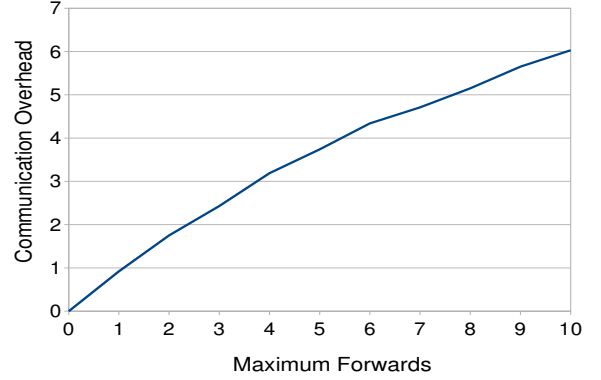


Figure 2: Percentage of messages added to the system

In addition, as the number of forwards increases, the redundant messages added into the system also increase. From Figure 2, we can see that such increase is almost linear. When the maximal number of forwards is 10, the amount of redundant messages in the system is six times more than original messages. Therefore, constraining the maximal number of forwards is crucial in reducing network bandwidth consumptions.

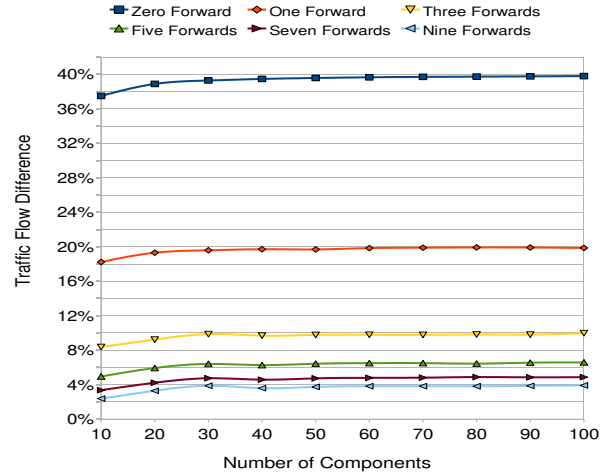


Figure 3: Traffic difference reduction under different system scales

The second set of experiments is to investigate the scalability of our approach. In particular, we create a system with total 10, 20, 30, ..., 100 components, respectively. There are two high traffic components in each group. In this experiment, we still assume that each of the high traffic components contributes 40% of the total messages. There are totally 10,000 messages communicated in the system initially. From Figure 3 we can see that the traffic difference between the highest traffic component and lowest traffic component decreases as the number of maximal forwards

increases. For example, for the system which has 100 components, the traffic difference drops to around 20% when the maximal forward is one, and reaches around 4% when the maximal forward is nine. The empirical data indicates that the approach scales well.

From the experiments presented above, it is clear that through message forwarding, we can effectively disguise critical components from attackers through preventing the attackers recognizing real traffic patterns. However, the simulation results also show that the high availability is at the cost of increased network traffic flow. Therefore, depending on the system run-time environment, we shall intelligently choose when and for how many times to execute message forwarding.

5. CONCLUSION

We have presented a novel approach to improve system availability when the system is under intentional attacks. It is to camouflage real messages and prevent attackers from correct traffic analysis that may reveal the location of more important software components and thus prolongs the time for attackers to comprise the system.

The theoretical analysis helps to decide an optimal number of forwards based on the traffic difference threshold that attackers use in identifying the traffic pattern. However, the message forwarding may not be necessary if the system's running environment is safe. For our future work, we will integrate intrusion detection techniques into our model and dynamically decide when and for how many times to execute message forwarding based on runtime risk and reduce unnecessary amount of redundant messages.

6. REFERENCES

- [1] R. Albert, H. Jeong, and A.-L. Barabasi. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, July 2000.
- [2] G. Andersson. Lpi performance of an adaptive frequency-hopping system in an hf interference environment. In *IEEE 4th International Symposium on Spread Spectrum Techniques and Applications*, volume 2, pages 903–907, 1996.
- [3] J. Arlat, K. Kanoun, and J.-C. Laprie. Dependability modeling and evaluation of software fault-tolerant systems. *IEEE Trans. Comput.*, 39(4):504–513, 1990.
- [4] R. Cohen, K. Erez, D. Ben-Avraham, and S. Havlin. Breakdown of the internet under intentional attack. *Phys. Rev. Lett.*, 86(16):3682–3685, Apr 2001.
- [5] D. E. Eckhardt and L. D. Lee. Fundamental differences in the reliability of n-modular redundancy and redundancy and n-version programming. *J. Syst. Softw.*, 8(4):313–318, 1988.
- [6] L. K. Gallos, P. Argyrakis, R. Cohen, and S. Havlin. Tolerance of scale-free networks: from friendly to intentional attack strategies. *Physica A*, 344:504–509, 2004.
- [7] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Hiding routing information. In *Proceedings of the First International Workshop on Information Hiding*, pages 137–150, London, UK, 1996. Springer-Verlag.
- [8] Y. Guan, X. Fu, D. Xuan, P. U. Shenoy, R. Bettati, and W. Zhao. Netcamo: camouflaging network traffic for qos-guaranteed mission critical applications. In *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, volume 31, pages 253–265, 2001.
- [9] K. Hausken and G. Levitin. Protection vs. redundancy in homogeneous parallel systems. In *Reliability Engineering & System Safety*, volume 93, pages 1444–1451, 2008.
- [10] K. Hausken and G. Levitin. False targets vs. redundancy in homogeneous parallel systems. In *Reliability Engineering & System Safety*, volume 94, pages 588–595, 2009.
- [11] K. Hausken and G. Levitin. Protection vs. false targets in series systems. In *Reliability Engineering & System Safety*, volume 94, pages 973–981, 2009.
- [12] K. Kwiat and S. Ren. A coordination model for improving software system attack-tolerance and survivability in open hostile environments. In *Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*, pages 394–402, 2006.
- [13] J. C. Laprie and B. Randell. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, 2004.
- [14] G. Levitin and K. Hausken. Redundancy vs. protection vs. false targets for systems under attack. In *IEEE Transactions on Reliability*, volume 58, pages 58–68, 2009.
- [15] B. Li, B. Saroj, and S. Musoke. How to update dependable secure computing systems from a survivability assessment perspective? In *Proceedings of the Third IEEE International Symposium on Dependable, Autonomic and Secure Computing*, pages 95–99, 2007.
- [16] M. Lienhardt, A. Schmitt, and J.-B. Stefani. Typing communicating component assemblages. In *GPCE '08: Proceedings of the 7th international conference on Generative programming and component engineering*, pages 125–136, New York, NY, USA, 2008. ACM.
- [17] R. Newman-Wolfe and B. Venkatraman. High level prevention of traffic analysis. In *Proceedings of Computer Security Applications Conference*, pages 102 – 109, 1991.
- [18] D. Powell. Failure mode assumptions and assumption coverage. *Twenty-Second International Symposium on Fault-Tolerant Computing*, pages 386–395, July 1992.
- [19] J. Wang and J. Bigham. Anomaly detection in the case of message oriented middleware. In *MidSec '08: Proceedings of the 2008 workshop on Middleware security*, pages 40–42, New York, USA, 2008. ACM.
- [20] J. Wu, H. Deng, Y. Tan, and D. Zhu. Vulnerability of complex networks under intentional attack with incomplete information. In *Journal of Physics A: Mathematical and Theoretical*, volume 40, pages 2665–2671, 2007.
- [21] S. Xiao and G. Xiao. On intentional attacks and protections in complex communication networks. *Global Telecommunications Conference*, pages 1 – 5, 2006.
- [22] D. Xuan, C. Li, D. Xuan, R. Bettati, and W. Zhao. Preventing traffic analysis for real-time communication networks. In *Proceedings of The IEEE Military Communication Conference (MILCOM) '99*, pages 744–750, 1999.